Server-Based Rendering of Large 3D Scenes for Mobile Devices Using G-Buffer Cube Maps

Juergen Doellner*
Hasso-Plattner-Institut,
University of Potsdam

Benjamin Hagedorn†
Hasso-Plattner-Institut,
University of Potsdam

Jan Klimke‡
Hasso-Plattner-Institut,
University of Potsdam

Abstract

Large virtual 3D scenes play a major role in growing number of applications, systems, and technologies to effectively communicate complex spatial information. Their web-based provision, in particular on mobile devices, represents a key challenge for system and application development. In contrast to approaches based on streaming 3D scene data to clients, our approach splits 3D rendering into two processes: A server process is responsible for real-time rendering of virtual panoramas, represented by G-buffer cube maps, for a requested camera setting. The client reconstruction process uses these cube maps to reconstruct the 3D scene and allows users to operate on and interact with that representation. The key properties of this approach include that (a) the complexity of transmitted data not depend on the 3D scene's complexity; (b) 3D rendering can take place within a controlled and a-priori known server environment; (c) crucial 3D model data never leaves the server environment; and (d) the clients can flexibly extend the 3D cube map viewer by adding both local 3D models and specialized 3D operations.

CR Categories: C.2.4 [Computer-Communication Networks]: Distributed Systems—Client/server, distributed applications I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics

Keywords: Web-Based 3D Rendering, Service-Oriented 3D Rendering, Remote 3D Rendering, Large 3D Scenes

1 Introduction

Large virtual 3D scenes are essential to systems, applications, and technologies. In general, these models represent a complex artifact or environment and allow users to visualize, explore, analyze, edit, and manage its components, i.e., they serve as tools to effectively communicate spatial information. With the rapidly growing demand for web-based and mobile applications, it becomes crucial for system and application developers to provide web-based, interactive access to large, virtual 3D scenes.

One fundamental challenge represents provision and delivery of 3D contents across networks. 3D Models are constantly getting more complex since they improve in terms of detail and realism. Although acceleration and compression techniques can reduce the 3D data volume, large-scale models are difficult to transfer and provide instantaneously for mobile devices. Another fundamental challenge is concerned with the complexity of 3D rendering algorithms. Even if we assume that 3D graphics capabilities of mobile devices will rapidly grow and, therefore, will be able to cope with complex 3D rendering tasks, complex 3D graphics systems are difficult to adapt to and deploy on mobile devices, are difficult to test and maintain, and lead to high energy consumption during their operation [Huang et al. 2009]. For that reason, we focus on a server-client communication that is independent from the 3D scene complexity and supports interactive and robust 3D visualization on the clients.

Figure 1: *Service-based rendering of a large 3D city model.*

A straightforward technical approach consists of streaming 3D scene data to clients that are responsible for managing and rendering the 3D scene data. The inherent limitations include that both data transmission and rendering performance directly depend on the 3D scene's complexity and the 3D graphics capabilities of the mobile device. Our approach uses a different strategy that relies on server-based 3D scene rendering and client-based 3D scene reconstruction based on virtual panoramas, technically represented by G-buffer cube maps (Fig. 2). This way, we can avoid streaming 3D scene data to clients and, therefore, decouple 3D scene complexity from data transmission complexity. In addition, the server can use advanced 3D rendering technology, while only moderate 3D graphics capabilities on the clients are required.

The server-based 3D rendering process is responsible for real-time rendering of large 3D scenes, generating G-buffer cube maps for a given camera setting; a G-buffer [Akenine-Möller et al. 2008] stores, e.g., RGB, depth, normal, and object-id data (Fig. 3). The G-buffer cube maps are streamed to the clients as texture data. The clients "reconstruct" the 3D scene based on image-based rendering using these cube maps. The interactive viewer autonomously operates on a local cube map cache and, therefore, allows for a stable and robust client-side user interaction.

The approach for server-based 3D rendering of large 3D scenes

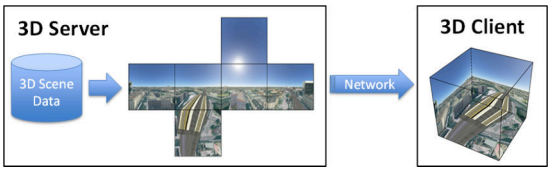

Figure 2: *Distributed 3D rendering schema.*

*e-mail:doellner@hpi.uni-potsdam.de
†e-mail:benjamin.hagedorn@hpi.uni-potsdam.de
‡e-mail:jan.klimke@hpi.uni-potsdam.de

Web3D 2012, Los Angeles, CA, August 4 – 5, 2012.
© 2012 ACM 978-1-4503-1432-9/12/0008 $15.00

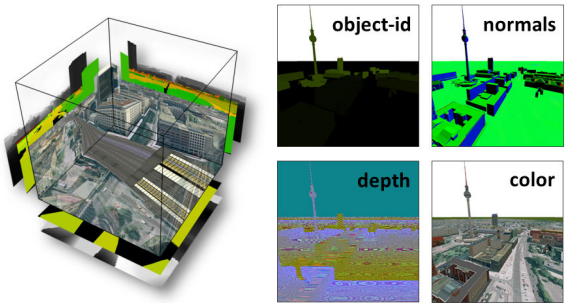

Figure 3: *G-buffer cube map. Left: Schema showing three of six cube map sides; right: four example G-buffers.*

is characterized as follows: (1) The complexity of G-buffer cube maps, which are transmitted between server and client, is independent from the 3D scene's complexity. (2) 3D rendering takes place within a controlled server environment. In particular, known and tested 3D graphics hardware facilitates the implementation of advanced 3D rendering techniques (e.g., multi-pass rendering, shader programming). (3) The service interface of the 3D Server is based on the evolving Web View Service standard proposa [Hagedorn 2010] for 3D portrayal of the in the Open Geospatial Consortium (OGC) in order to provide a reusable, interoperable 3D rendering server. (4) Web-based, interactive usage is provided, while the original 3D contents are kept protected: 3D scene data is used by the server, but never leaves the server environment. (5) The clients can render and integrate own, e.g., locally stored or streamed, 3D models as depth information is provided by the G-buffer cube map.

In a case study, we applied our prototypical implementation to disseminate and interactively visualize large-scale virtual 3D city models. We show the potentials of our approach by an iPhone/iPad app that allows users to freely navigate through and interact with these models.

2 Related Work

Many software architectures, systems, and frameworks for remote 3D visualization are based on streaming image and video contents, e.g., streaming frame sequences based on MPEG [Lamberti and Sanna 2007]; Paravati et al. [Paravati et al.] introduce a scalable architecture for the delivery of shared 3D visualizations to heterogeneous mobile and desktop devices based on sharing a common view generated by a 3D rendering server. Our approach is based on delivering virtual panoramas instead of frame sequences and takes advantage of similar load balancing mechanisms. Another main category of solutions is based on streaming 3D scene data (e.g., encoded by X3D, VRML) to clients that locally perform 3D rendering (e.g., GoogleEarth, Bing Maps 3D). A hybrid variant introduced by [Koller et al. 2004] uses a 3D client visualizing a low-resolution 3D model version, which is refined by a high-resolution view generated by the server; this way, the original 3D contents are kept protected. In contrast, we focus on an explicit separation of 3D model data (server) and derived image-based representations (client).

For large 3D scenes, a multitude of real-time rendering techniques exist such as *out-of-core rendering* (OOC) [Gobbetti et al. 2008]. Complementary, *image-based modeling and rendering* (IBMR) [Shum et al. 2007] offers techniques for remote 3D rendering of complex scenes, for instance, client-side warping of color and depth images retrieved from a remote server [Chang and Ger 2002]. For it, the depth mesh of a view is decomposed according to its depth profile into a small set of textured polygons; the 2.5D representation is used by the 3D client and facilitating to render "plausibly distorted views of the scene at high frame rates as long as the viewing position does not change too much before the next frame arrives from the server" [Li et al. 2011]. Our approach builds upon IBMR techniques and extends these to cube maps.

3 Software Architecture

In the proposed software architecture (Fig. 4), we distinguish between the following processes: a) The *server-side 3D rendering* generates G-buffer cube maps of 3D scenes for client requests and handles the streaming of the results. b) The *client-side 3D rendering* process receives requested G-buffer cube maps, reconstructs the 3D scene based on the cube map information, and interactively visualizes the virtual panorama. The client can additionally provide application-specific 3D functionality, using the cube map as base scene.

3.1 3D Server

The 3D server, technically, performs three parallel processes that communicate by shared data repositories. The *3D request process* manages requests for G-buffer cube maps sent by the clients. It can sort these requests according to their priority, and dispatch these requests to available *3D rendering processes* that is responsible for image synthesis and scene management. It accesses the 3D scene database, constructs and optimizes the corresponding scene graph, and implements the core 3D rendering algorithms. As intermediate output, this process generates G-buffer cube maps, stored as 2D texture data. These outputs can be further processed by another image-based rendering stage, for instance, to stylize or finish these renderings. For the post-processing, a dedicated image processing system can be used or it can form part of the core 3D rendering process. One reason to separate core 3D rendering from cube map post-processing is that this allows us to deploy the post-processing on a different server, i.e., it improves the scalability of the architecture. Finally, a post-processed cube map is stored in the cube map server repository. As the final stage of request processing, the *streaming process* is responsible for monitoring the cube map repository, i.e., compression and streaming of finalized cube maps.

3.2 3D Client

The 3D client application provides a graphical user interface and is responsible for interactive display of and user interaction with the reconstructed 3D scene, including the request management. Therefore, G-buffer cube maps generated by the 3D Server are transmitted to the client that manages these textures using a local cube map cache. To render a reconstructed scene, the client builds a local scene graph that primarily consists of cube map geometry and textures. The client implementation can be based on, for example,

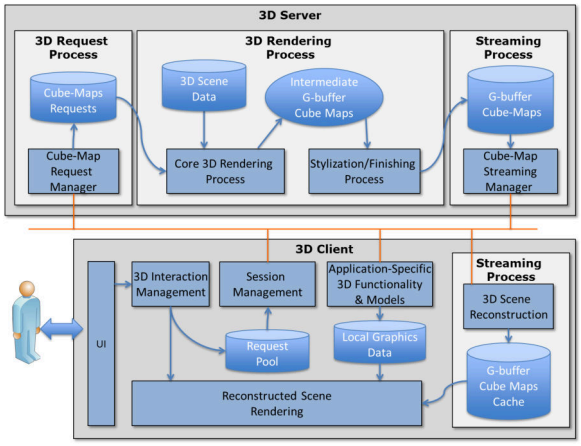

Figure 4: *Architecture of the service-based 3D rendering system.*

OpenGL ES or WebGL. Generally, this functionality can be implemented by fundamental geometry and texturing operations as provided by OpenGL ES. In particular, no advanced shader capabilities are needed.

The client application implements the management of 3D user interaction. It takes care of multi-touch inputs that occur on top of the graphics canvas. It interprets these events in terms of camera movements, which are delegated to the reconstructed scene rendering process. As part of this functionality, it is responsible for requesting new cube maps if the camera settings significantly change. The *session management* handles the connection between client and server. According to the needs of a specific application scenario, this can include individual profiles and settings on a per-user basis. The graphical user interface for a client application is implemented as a device and operation-system specific module, providing a native user experience for the designed target platform (e.g., iPhone App, Android App, or WebGL-based script).

4 G-Buffer Cube Maps

G-buffer cube maps are the core elements used to discretize complex 3D scenes for streaming. They combine two separate concepts, cube maps and G-buffers.

Cube mapping represents an efficient, real-time technique for environment mapping [Akenine-Möller et al. 2008]. In a sense, a cube map denotes a virtual cube-shaped panorama, taken around the cube center. The cube map can be encoded by 2D textures of same size and format, typically stored into an array texture. The generation process is accelerated by a number of extension in OpenGL as array textures can be bound to render targets of framebuffer objects. A cube map can be generated in a single rendering pass, i.e., the scene is simultaneously rendered into the six different rendering targets, provided that scene management, culling, and acceleration techniques are adapted; otherwise six rendering passes are required.

The G-buffer represents a key concept and technique for real-time enabled implementation of image-based rendering algorithms. Beside standard G-Buffer types such as color, depth, normal or stencil buffers, we use G-Buffers to store other 3D scene related information, such as world coordinates, surface parameterization, object identifiers, or application-specific attributes (i.e., thematic data), on a per-fragment basis — these values can be derived using adapted shader programs during the rendering process. Further, a fundamental benefit of the G-buffer cube maps is that both the 3D server (in the post-processing step) as well as the 3D client (as part of its rendering process) can apply image-based rendering techniques, such as object or silhouettes highlighting , deferred lighting, or non-photorealistic rendering, to enhance depictions.

As the sides of a cube map encompass the whole scene for the given view point, the *G-buffer cube map* represents a discretized, omni-directional approximation of the 3D scene that can be efficiently stored by fixed-size OpenGL array textures. The resolution and the selection of G-buffer types depends on the client requirements; e.g., a minimal version requires RGB and depth layers. To support interactive client-side 3D object selection, object-id and object-type layers need to be included.

To reduce streaming bandwidth and volume, the array textures can be compressed (e.g., JPEG). The optimal choice of a compression technique also depends on the numeric type and precision requirements of a layer. For example, the object-id layer needs to be lossless compressed because value interpolation is not feasible.

As the 3D client needs to process the streamed data as textures, texture compression formats offer the advantage that client-side in-memory decompression is not required provided that server and client agree on the available hardware support. Unfortunately, compressed texture formats (e.g., ETC, S3TC) are currently not universally supported; recent advances in OpenGL and OpenGL ES suggest to apply RGTC or BPTC variants as well as vendor-specific formats (e.g., PVRTC on iPhone).

5 Server-Side 3D Rendering

The 3D server encapsulates the generation of G-buffer cube maps. A cube map request, sent by clients, is defined by (1) the virtual camera specification, (2) a list of requested image layers and their encodings, (3) a list of requested model layers to be rendered, and (4) the styling and post-processing specifications.

To reduce network communication overhead and to allow for internal server-side optimization, a client can request several cube maps each consisting of several image layers with one operation call. The *WVS Endpoint* parses a client request according to the WVS specification [Hagedorn 2010] and calls the appropriate operation handler, passes it to the *Render Master*.

5.1 Parallel Processing of Requests

The *Render Master* process distributes the workload related to incoming requests to available processing resources, denoted by *Render Worker*s. The Render Master and *Web Server* perform process synchronization to manage shared resources. The major goals include to optimize resource utilization, maximize throughput, and minimize response times.

The *Render Master* queues *WVS GetView* requests, processes them and assigns *Tasks* to free *Render Workers*. When processing a request, it is split into a set of *Tasks*; each Render Worker can process exactly one task at a time. Tasks can be split into subtasks based on *sort-first decomposition* For it, the view plane is split into a set of 2D tiles.

A *Render Worker* has its own thread of execution and an associated GPU for processing tasks. Either it executes on the same CPU as the *Render Master*, on a different CPU on the same computer, or a different computer connected via a network. Each requested view consists of one or more G-buffer layer. Tasks are organized in a way so that post processing view is done on the same worker that generated the view. This has several advantages: First, not only the rendering and G-buffer generation but also post-processing can execute concurrently on workers. In addition, generated G-buffers still reside in GPU memory and can be used in-place for post-processing. Finally, temporary G-buffers created only for post-processing effects never have to be transferred from GPU memory.

The creation of G-buffer layers of a single cube map side is never distributed to different tasks as multiple G-buffer layers for a given view can be most efficiently created on the same worker in one rendering pass using *multiple rendering targets* in one OpenGL context. Furthermore, since OOC rendering techniques must be applied, the data required for generating a G-buffer layer must be transferred from external storage prior to rendering. Last, but not least, post-processing may want to use G-buffer layers, directly available to the GPU.

6 3D Scene Reconstruction

The 3D client basically displays the textured cube map geometry, with the initial camera view point in the cube center. The minimal graphics requirements are low: the six-sided cube requires 12 triangles and corresponding six 2D textures (e.g., 512×512 texel).

The user navigation and interaction operates always on the most recent G-buffer cube map received. If the camera is moved or zoomed, a request for a new cube map is filed, and the current cube map is marked as outdated and an updated one is fetched from the server. Still, navigation and interaction can operate on the current cube map, independently of the provision of the requested new cube map. The artifacts that tend to occur include insufficient image resolution (i.e., blurring effects) and incorrect 3D object visibility.

The client-side 3D rendering process can be enhanced by various image-based rendering techniques. Most importantly, a cube map's depth layer can be alternatively encoded as a depth mesh [Pajarola et al. 2004], which represents a triangulation of the depth image. This way, depth meshes are "directly enabling fast intermediate-view rendering" [Farin et al. 2007], using the depth mesh instead of the cube map geometry; efficient algorithm for compression and meshing exist [Sarkis et al. 2010]. The client switches to a depth mesh scene representation as soon as the position of the virtual camera changes. However, if a depth mesh is not displayed from the reference view, usually holes or rubber-sheets become visible, which can be avoided by requesting and rendering additional depth meshes at the cost of performance and utilized network bandwidth.

7 Case Study

We have built the 3D server system based on C++, OpenGL and OpenSceneGraph; clients have been developed for iPhone, iPad, and WebGL-enabled browsers. As example, we have used the large-scale, virtual 3D city model of Berlin (Fig. 1) covering approx. 890 km^2 and including over 550.000 buildings (350 buildings modeled in high detail), with nearly 4.000.000 single 2D facade textures. Further geo-referenced map and feature data can be accessed on demand by WMS or WFS (e.g., topographic map or public transport networks). There are two major application scenarios, both related to city information and marketing: a) In face-to-face meetings, representatives of various departments want to use their tablets to interactively explore and analyze using the virtual 3D city model. Typically, they focus on the specific interests of customers that are present, i.e., they have to have interactive, direct access according to the course of the conversation and b) Presentations of selected areas of interests, for example, on exhibitions or trade fairs, are used to communicate potentials, or concepts of future real-estate projects.

In our prototype implementation, we integrated a specialized variant to include 2D and 3D scene data on demand. For it, the virtual 3D city model is enhanced and can be reconfigured by different geodata such as topographic maps, standard land value maps, transportation and infrastructure networks, architectural models, etc. The data can be accessed, e.g., by OGC WMS and WFS, which are accessed via the 3D server process. The activation and configuration can be controlled by the session management of the client.

To measure the speed of the 3D rendering process and required bandwidth, we have set up a service consumer (running on a PC) to stress the 3D rendering process. The experiment was performed on a dedicated server system (Ubuntu, 3.4GHz quad core, 16 GB RAM, GeForce GTX 560 Ti) in an intranet environment. The service consumer received images at an average rate of 4.6 G-buffer cube maps per second for an image resolution of 512x512 and of 2.1 1024x1024. The memory size of cube maps while simulating navigating through the 3D scene (resolution 512x512) required on average 78.95 kbytes for color (JPEG), 199.63 kbytes for depth (PNG), and 9.56 kbytes for object-ids (PNG).

The implemented iOS application running on an iPad2 worked constantly at interactive frame rates ($>=$ 60 fps) during camera rotation (i.e., pure rendering of the cube map), flight-mode navigation (using regular depth mesh triangulation). ,and walk-mode navigation (rendering up to 12 depth meshes). It is obvious that today's mobile devices with 3D graphics support are able to render the G-buffer cube maps in real-time; there is even a large potential for advanced client-side image-based rendering techniques.

The service-oriented 3D rendering approach could significantly reduce cost and time intensive efforts, resulting from, e.g., physically moving the former 3D desktop system (3D and storage hardware) to remote locations. For example, instead of using a single view on a large screen, participants can have specific views on their tablets.

8 Conclusions and Future Work

Using our approach, the complexity of 3D scene data and transmitted data is decoupled. Client GPU, memory, and energy resources are efficiently used by G-buffer cube map processing. The multiple layers of the G-buffer cube maps allow clients to implement advanced 3D operations and integrate additional 3D models. The rendering of the 3D scene is in the scope of the server and, therefore, can be implemented for a-priori known GPU environments, facilitating a cross-platform, interoperable use of the rendering service. 3D model data, in addition, is fully kept on the server, thus providing a high degree of protection for 3D contents. The interactive client display operates in robust way under unstable and network throughput fluctuations using a cube map cache. The case study on large virtual 3D city models indicates that users realize the client application as "true 3D" application, and they get rapidly familiar with the refining delays while new, more appropriate cube maps are requested, above all, because the interaction never gets stuck.

In future work, we will investigate how we can take advantage of user navigation prediction to optimize the cube map request management. Furthermore, the client 3D rendering demands for specialized image-based techniques to use the potential of multiple layers encoded by G-buffer cube maps. Currently, the concepts of this approach have been filed into an ongoing standardization process of the OGC.

References

AKENINE-MÖLLER, T., HAINES, E., AND HOFFMAN, N. 2008. *Real-Time Rendering*, 3 ed. A. K. Peters, Ltd.

CHANG, C.-F., AND GER, S.-H. 2002. Enhancing 3D Graphics on Mobile Devices by Image-Based Rendering. In *Proc. of PCM 2002*.

FARIN, D., PEERLINGS, R., AND DE WITH, P. H. N. 2007. Depth-Image Representation Employing Meshes for Intermediate-View Rendering and Coding. In *3DTV-CON 2007 - Capture, Transmission and Display of 3D Video*.

GOBBETTI, E., KASIK, D., AND YOON, S.-E. 2008. Technical Strategies for Massive Model Visualization. In *Proc. of SPM '08*, 405–415.

HAGEDORN, B., Ed. 2010. *Web View Service Discussion Paper, v0.6.0*.

HUANG, S., XIAO, S., AND FENG, W. 2009. On the Energy Efficiency of Graphics Processing Units for Scientific Computing. In *Proc. of IPDPS '09*, 1–8.

KOLLER, D., TURITZIN, M., LEVOY, M., AND TARINI, M. 2004. Protected Interactive 3D Graphics via Remote Rendering. In *SIGGRAPH 04*, vol. 1, 695–703.

LAMBERTI, F., AND SANNA, A. 2007. A Streaming-Based Solution for Remote Visualization of 3D Graphics on Mobile Devices. *IEEE TVCG 13*, 2, 247–260.

LI, M., SCHMITZ, A., AND KOBBELT, L. 2011. Pseudo-Immersive Real-Time Display of 3D Scenes on Mobile Devices. In *3DIMPVT*.

PAJAROLA, R., SAINZ, M., AND MENG, Y. 2004. DMesh: Fast Depth-Image Meshing And Warping. *IJIG 4*, 4, 653–681.

PARAVATI, G., SANNA, A., LAMBERTI, F., AND CIMINIERA, L. An Open and Scalable Architecture for Delivering 3D Shared Visualization Services to Heterogeneous Devices.

SARKIS, M., ZIA, W., AND DIEPOLD, K. 2010. *Fast Depth Map Compression and Meshing with Compressed Tritree*, vol. 5995/2010 of *LNCS*.

SHUM, H.-Y., CHAN, S.-C., AND KANG, S. B. 2007. *Image-Based Rendering*.